2023년 K-ium
의료인공지능경진대회

키움화이팅

김수인

곽엘림

박지은

박민영

한규원

하민세

2023/07/13

목차 및 요약

dataset

- train, validation을 위한 data 설명

class 나눈 기준

- data label의 분리 방법

image preprocessing

- 일부 전처리가 필요한 raw image 들의 설명 및 처리 방법

모델 예측 방식

- 모델의 예측 pipeline

model

- binary / multi label classfication을 위한 모델 설명

etc

- 그밖의 했던 시도들

Dataset

크게 Aneurysm 예측을 위한 Dataset과 각 위치들을 예측하기위한 Dataset으로 나누고, 두 Dataset을 각각 anterior(carotid), posterior(vertebral)로 나눔. 총 4개의 Dataset이 존재

위치 Dataset - multilabel classification

1) 사진에서 I,V를 기준으로 전/후방 image path들을 나눔

2) train.csv에서 L/R, I/V를 기준으로 각각에 맞는 Column들을 할당한 새 Dataframe 생성

```
L_anterior = ['Index', 'L_ICA', 'L_AntChor', 'L_ACA', 'L_ACOM', 'L_MCA']
R_anterior = ['Index', 'R_ICA', 'R_AntChor', 'R_ACA', 'R_ACOM', 'R_MCA']
L_posterior = ['Index', 'L_VA', 'L_PICA', 'L_SCA', 'BA', 'L_PCA', 'L_PCOM']
R_posterior = ['Index', 'R_VA', 'R_PICA', 'R_SCA', 'BA', 'R_PCA', 'R_PCOM']
```

3) train.csv 크기(len)의 4배, 칼럼은 동일, 새로운 Dataframe anterior, posterior 생성 (=> 한 명당 8장의 이미지인 데이터를 전/후방으로 나누었기에 크기가 4배임)

```
L_anterior_df.rename(columns={'L_ICA':'ICA', 'L_AntChor':'AntChor', 'L_ACA':'ACA',
'L_ACOM':'ACOM', 'L_MCA':'MCA'}, inplace=True)
R_anterior_df.rename(columns={'R_ICA':'ICA', 'R_AntChor':'AntChor', 'R_ACA':'ACA',
'R_ACOM':'ACOM', 'R_MCA':'MCA'}, inplace=True)
```

4) anterior/posterior에 각 이미지에서 보이는 레이블에 해당하는 값들을 적절한 간격으로 L/R_atnterior, L/R_posterior의 값으로 할당

```
L_posterior_df.rename(columns={'L_VA':'VA', 'L_PICA':'PICA', 'L_SCA':'SCA', 'L_PCA':'PCA',
'L_PCOM':'PCOM'}, inplace=True)
R_posterior_df.rename(columns={'R_VA':'VA', 'R_PICA':'PICA', 'R_SCA':'SCA', 'R_PCA':'PCA',
'R_PCOM':'PCOM'}, inplace=True)

anterior = pd.DataFrame(index=range(4*len(data)), columns=L_anterior_df.columns)
anterior.iloc[::4, :] = L_anterior_df.values
anterior.iloc[2::4, :] = R_anterior_df.values
anterior.fillna(method='ffill', inplace=True)
anterior['sum'] = anterior.iloc[:, 1:].sum(axis=1)
```

```
anterior['path'] = anterior_path


posterior = pd.DataFrame(index=range(4*len(data)), columns=L_posterior_df.columns)
posterior.iloc[::4, :] = L_posterior_df.values
posterior.iloc[2::4, :] = R_posterior_df.values
posterior.fillna(method='ffill', inplace=True)
posterior['sum'] = posterior.iloc[:, 1:].sum(axis=1)
posterior['path'] = posterior_path
```

*anterior과 posterior의 레이블에서 L/R을 하나로 통합했지만, 각각 같은 값을 가지는 것이 아닌 사진의 L/R에 따라서 같은 레이블이라도 1인 경우와 0인 경우가 따로 나타나게 됨 (추론 시 L/R 레이블을 다시 생성)

Aneurysm dataset- binary classification

1) 만들어진 anterior과 posterior을 기준으로 각 행에 대하여 1이 한 개라도 있으면 'sum'이라는 label에 1로 할당, 그 외엔 0

```
binary_anterior = anterior.copy()
binary_anterior.drop(['ICA', 'AntChor', 'ACA', 'ACOM', 'MCA'], axis=1, inplace=True)
binary_anterior['sum'] = anterior.iloc[:, 1:].sum(axis=1)
binary_anterior['sum'] = binary_anterior['sum'].map(lambda x: 1 if x > 0 else 0)


binary_posterior = posterior.copy()
binary_posterior.drop(['VA' ,'PICA' ,'SCA'  ,'BA' ,'PCA'  ,'PCOM'], axis=1, inplace=True)
binary_posterior['sum'] = posterior.iloc[:, 1:].sum(axis=1)
binary_posterior['sum'] = binary_posterior['sum'].map(lambda x: 1 if x > 0 else 0)
```

학습에 사용되는 데이터는 binary classification을 위한 binary_anterior, binary_posterior과 multilabel classifcation을 위한 anterior, posterior => 총 4개

Class 나눈 기준

한 명당 8장의 사진 중, 각각의 사진이 train.csv의 레이블에 해당하는 모든 위치들을 볼 수 없다고 판단하였음. 조사 결과, 뇌 혈액 순환은 전/후방에 따라 ICA/VA로 나누어 진다는 사실을 알게 되었고, 이에 따라 전/후방에 위치한 혈관으로 레이블을 나눔.

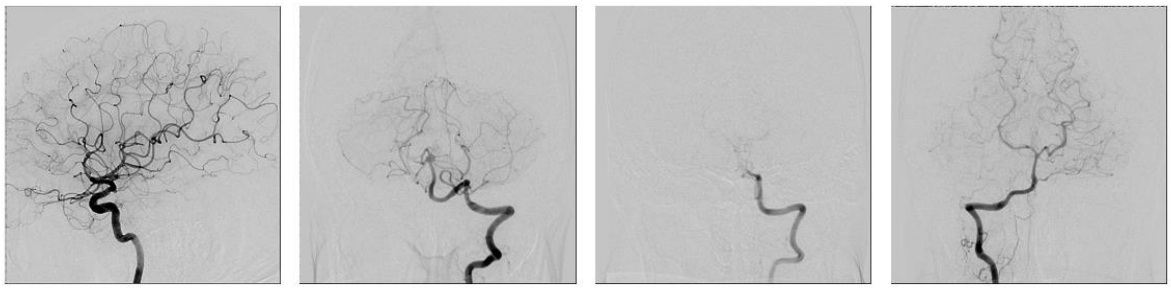
ICA: ICA, AntChor, ACA, ACOM, MCA

VA : VA, PICA, SCA, BA, PCA, PCOM

Image preprocessing

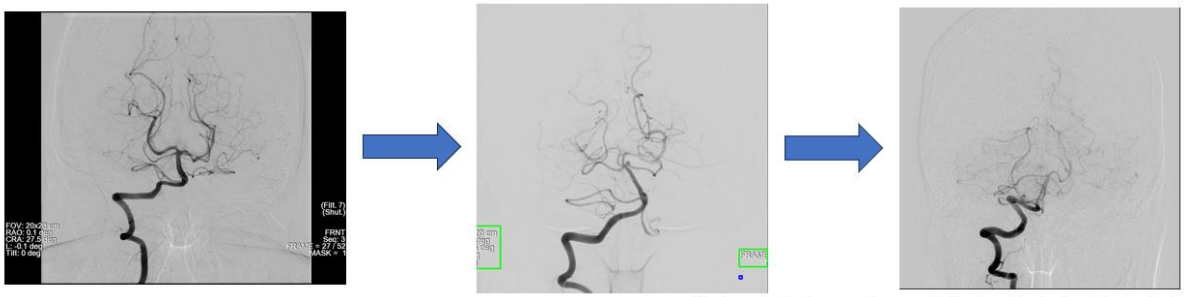
train dataset은 아래와 같이 3가지 타입으로 나뉘어져 있었으며, 여백과 글자가 없는 경우를 제외한 나머지는 이미지 전처리를 해줬음.

- **여백과 글자가 없는 경우**



- **글자와 여백이 모두 존재하는 경우**

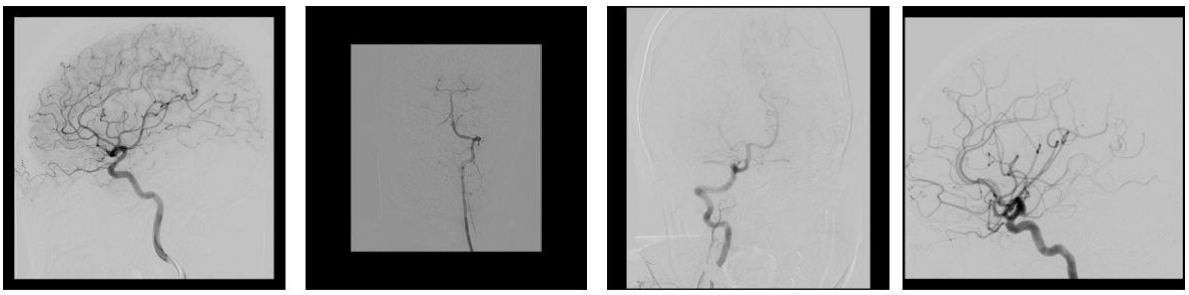
픽셀값을 이용하여 여백을 모두 지워주고 난 뒤, 글자가 모두 같은 위치에 있었기 때문에 고정좌표로 두 번째 사진과 같이 초록색 영역을 설정함. 이후 사진에 맞는 배경색을 가지고 오기 위해서 파란색 영역의 픽셀값을 평균내어 초록색 영역을 덮음.



해당 이미지는 같은 위치의 사진이 아닙니다

- **여백이 있는 경우**

이미지마다 여백의 크기가 달라, 이미지의 center를 기준으로 horizontal line과 vertical line을 슬라이스 한 뒤, 반복문을 돌려 gray color가 들어오기 시작하는 좌표를 찾아 이미지를 자름



모델 예측 방식

1) Binary classification으로 Aneurysm을 예측하고, 임계점을 넘어 가면 그 행(row)는 위치 label을 모두 0으로 대체 - 임계점은 train.csv에 대한 Aneurysm의 예측값의 중앙값
2) 임계점을 넘는 행은 Multi label classification으로 각 위치에 대한 값을 예측하고, 임계점을 두어 1 또는 0으로 대체 - 임계점은 train.csv에 대한 각 위치 예측값의 90백분위수들 (21개)

* Aneurysm 예측값의 형태는 9016x1 (한 사람 당 8장 이미지에 대한 모든 Aneurysm 예측) 이므로, anterior, posterior 각각 4장에 대하여 평균 2개를 구하고, 그 중 가장 큰 값으로 Aneurysm 값 결정

Model

1. **binary_classification - anterior/posterior 같은 모델 사용**

   * MedNet (resnet18) - 여러 medical, gray-scale로 학습된 가중치를 사용

   - https://huggingface.co/TencentMedicalNet/MedicalNet-Resnet10/blob/main/resnet_10.pt

```
class MedNet_resnet18(nn.Module):
  def __init__(self, in_dim, out_dim, weights_path='/content/drive/MyDrive/resnet_18.pth'):
    super(MedNet_resnet18, self).__init__()
    self.backbone = models.resnet18(weights=torch.load(weights_path))
    self.backbone.conv1 = nn.Conv2d(in_dim, 64, kernel_size=(7, 7),
                                    stride=(2, 2), padding=(3, 3),
                                    bias=False)
    self.backbone.fc = nn.Sequential(
        nn.Linear(512, 256),
        nn.Dropout(0.3),
        nn.ReLU(),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Linear(128,16),
        nn.ReLU(),
        nn.Linear(16, 1)
    )

  def forward(self, x):
    x = self.backbone(x)
    x = F.sigmoid(x)
    return x
```

   * Loss : BCELoss

   * augmentation : resize, normalize

   * 이미지에 cv2.bitwise_not, cv2.medianBlur를 적용

2. **multi label classification - 전/후방 다른 모델 사용**

   * SwinNet - timm 라이브러리 pretrained swinv2_cr_tiny_ns_224 사용 (anterior dataset)

```
import timm

class SwinNet(nn.Module):
    def __init__(self, out_features, inp_channels=1, pretrained=True):
        super(SwinNet, self).__init__()
        self.model = timm.create_model('swinv2_cr_tiny_ns_224', pretrained=True,
                                       in_chans=inp_channels)
        self.classifier = nn.Linear(1000, out_features)

    def forward(self, x):
        x = self.model(x)
        x = self.classifier(x)
        x = F.sigmoid(x)
        return x
```

* ResNet18 - torchvision pretrained resnet18 사용(posterior dataset)

```
class ResNet18(nn.Module):
    def __init__(self, output_dim):
        super(ResNet18, self).__init__()
        self.conv = nn.Conv2d(1, 3, kernel_size=(1, 1), stride=(1, 1), bias=False)
        self.backbone = models.resnet18(pretrained=True)
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(1000, 100),
            nn.ReLU(),
            nn.Linear(100, 10),
            nn.ReLU(),
            nn.Linear(10, output_dim)
        )

    def forward(self, x):
        x = self.conv(x)
        x = self.backbone(x)
        x = self.classifier(x)
        x = F.sigmoid(x)

        return x
```

* Loss : Asymmetric Loss
    - https://github.com/Alibaba-MIIL/ASL/blob/main/src/loss_functions/losses.py

* augmentation : resize, normalize

etc

- 전처리

1. Rule-base Crop을 이용한 학습 이미지 전처리

방식 : 동맥류 양성 레이블에 따라 이미지를 해당 위치 기준으로 Rule-base Crop해 학습하고자 함

이유 : 고화질의 이미지를 Resize 하는 과정에서 이미지의 화질이 저하되고, 이는 Feature loss로 이어질 것이라 판단

결과/분석 : 뇌동맥류가 특정위치에 2개 이상 있더라도 위치정보에는1로 표기되어 있어 정확한 위치를 특정하지 못해 진행하지 못함

2. 이미지 데이터를 위치/각도/방향에 따라 분할한 뒤 각각 학습

방식 : 위치, 각도, 방향에 따라 이미지셋을 분할 한 뒤, 그 이미지에서만 볼 수 있는 레이블만 이용해 따로따로 학습하고자 함(Ex : LI-A 이미지 + LI-A에서만 보이는 레이블만 학습한 모델 등).

이유 : 위치, 각도, 방향이 같은 이미지들은 비슷한 형태를 띄고, 판단해야 할 레이블 수도 적어 학습에 유리할 것으로 판단. weighted sampler + weightes classes 모두 이용

결과/분석 : 데이터 분할 시 양/음성 데이터 간 불균형에 영향을 더 많이 받아 데이터가 적은 클래스의 경우 예측을 제대로 하지 못함. weight 기법들로인한 정확도 증가 미미

Self-Supervised Learning

1. Autoencoder - image classification 연결

방식 : 데이터셋 label에 있어 불균형이 있었는데, 불균형에 영향을 적게 받으며 이미지의 특성을 학습하는 autoencoder의 특성을 이용하였다.

원리 : 입력을 저차원의 특성 벡터로 인코딩한다. 그 후 인코더에서 생성된 잠재 표현을 사용하여 원본 데이터를 복구하는데, 이렇게 재구성한 이미지가 원본 이미지와 비슷하도록 loss를 줄여나간다. 그 결과로 재구성된 이미지는 원본이미지와 유사하되, 특성을 학습하게 되는 원리이다.

200번의 epoch를 돌린 후, path에 있던 이미지들을 autoencoder의 pt파일을 통과하도록 하여 학습된 특성을 강조하도록 하였고, 그 후 binary classification으로 label이 존재하는 supervised learning을 진행하였다.

supervised learning : 지도학습을 할 때는 처음에는 전체 데이터에 대해 진행하였는데, loss가 떠도 불균형이 너무 심했기에 의미가 없었고, 두 가지 시도를 하였다.

label의 개수를 어느정도 맞추기 위해 많은 개수를 가지는 label을 다운샘플링(랜덤으로 고르고 나머지는 버림)으로 했지만 데이터의 개수가 너무 작았음, weighted sampling을 하였을 때도 다운샘플링과 결과는 비슷했다.

2. Contrastive Learning

- 이미지 데이터셋에서 라벨 불균형을 극복하기 위해 Contrastive Learning을 적용하였다. 이미지는 사전 훈련된 ResNet50 모델을 통해 특성을 추출하고, 이를 기반으로 같은 클래스의 이미지는 가까운 임베딩 공간으로, 다른 클래스의 이미지는 멀리 놓음으로써 유사성을 학습한다. 이 과정은 라벨 불균형이 덜 영향을 미치며, 각 이미지의 특성을 더 정확하게 반영할 수 있다.

ResNet18(fc-layer → conv layer)

일반적인 image classification을 하는 CNN모델들은 last layer가 fully connected layer(fc layer)로 이루어져 있음. fc layer는 픽셀의 위치 정보가 사라져 segment에 취약함. fc layer를 conv-layer로 바꿔 전처리와 augmentation을 하지 않은 데이터셋을 이용하여, ResNet18 모델을 학습시킴. 결과는 모든 에폭에서 같은 53.68%(f1 score)가 나옴.

```
[ ]  class MyCNN(nn.Module):
         def __init__(self, num_classes):
             super(MyCNN, self).__init__()
             self.features = nn.Sequential(
                 nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
                 nn.ReLU(inplace=True),
                 nn.MaxPool2d(kernel_size=2, stride=2),
                 nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
                 nn.ReLU(inplace=True),
                 nn.MaxPool2d(kernel_size=2, stride=2),
             )
             self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
             self.fc = nn.Linear(128, num_classes)

         def forward(self, x):
             x = self.features(x)
             x = self.avgpool(x)
             x = torch.flatten(x, 1)
             x = self.fc(x)
             return x

     model = MyCNN(num_classes=1)
     criterion = nn.BCEWithLogitsLoss()
     optimizer = optim.Adam(model.parameters(), lr=0.001)
```

DenseNet

DenseNet은 ResNet의 skip connection을 업그레이드 시킨 버전으로, 주어진 이미지의 feature를 파악하는 것이 가장 중요하다고 생각함. feature extraction 부분에서 성능이 높은 전처리와 augmentation을 하지 않은 데이터셋을 이용하여, DenseNet을 학습시킴.

```
class DenseNet(nn.Module):
    def __init__(self, num_classes):
        super(DenseNet, self).__init__()
        self.densenet = models.densenet121(pretrained=True)
        num_features = self.densenet.classifier.in_features
        self.densenet.classifier = nn.Linear(num_features, num_classes)

    def forward(self, x):
        return self.densenet(x)
```

Grad-Cam

여러 ResNet, DenseNet, VGG16 등 여러 CNN 모델을 돌려 보았지만, 이 모델이 이미지의 어떤 부분을 토대로 이러한 판단을 내렸는지 알 수 없었음. 또한, image augmentation을 하기 위해서는 이미지의 어떠한 부분이 중요한 지 알아야 했음. 설명 가능한 인공지능(eXplainable AI)의 기법 중 하나인 Grad-Cam을

적용하여 이미지 분석을 시도함. 전처리를 통해 글자와 여백을 제거하고 난 이미지를 이용하였고, augmentation은 하지 않았음.

ResNet18모델이 판단의 기준으로 이용한 부분을 히트맵으로 볼 수 있었으며, 모델의 정확도나 정답을 모르는 상태에서 Grad-Cam을 적용하였기 때문에 결과가 나왔지만 정확한 분석을 할 수 없었음.